

# On the Formal Specification of Regulatory Compliance: A Comparative Analysis

Amal Elgammal<sup>1</sup>, Oktay Turetken, Willem-Jan van den Heuvel, Mike Papazoglou

European Research Institute in Service Science (ERISS), Tilburg University,  
Tilburg, the Netherlands  
{a.f.s.a.elgammal, o.turetken, w.j.a.m.vdnheuvel, m.p.papazoglou}@uvt.nl

**Abstract.** Today's business environment demands a high rate of compliance of service-enabled business processes with which enterprises are required to cope. Thus, a comprehensive compliance management framework is required such that compliance management must crosscut all the stages of the complete business process lifecycle, starting from the very early stages of business process design. Formalizing compliance requirements based on a formal foundation of an expressive logical language enables the application of associated verification and analysis tools to ensure the compliance. In this paper, we have conducted a comparative analysis between three languages that can be used as the formal foundation of business process compliance requirements, focusing on *design-time* phase. Two main families of languages have been identified, which are: the temporal and deontic families of logic. In particular, we have considered LTL, CTL and FCL. The comparative analysis is based on the capabilities and limitations of each language and a set of required identified features.

**Keywords:** Compliance requirements specifications, linear temporal logic, Regulatory compliance, computational tree logic, formal contract language

## 1 Introduction

Today's business climate demands a high rate of compliance of business processes with which Information Technology (IT)-minded organizations are required to cope. Compliance regulations, such as Basel II, Sarbanes-Oxley and others require all organizations to review their business processes and service-enabled applications, and ensure that they meet the compliance constraints set so forth in the legislation.

Compliance is mainly ensuring that business processes, operations and practices are in accordance with a prescribed and/or agreed on set of norms [1]. A compliance

---

<sup>1</sup> This work is a part of the research project "COMPAS: Compliance-driven Models, Languages and Architectures for Services", which is funded by the European commission, funding reference FP7-215175.

constraint (requirement) is any explicitly stated rule or regulation that prescribes any aspect of an internal or cross-organizational business process.

SOA is an integration framework for connecting loosely coupled software modules into on-demand business processes. BPs form the foundation for SOAs and require that multiple steps occur between physically independent yet logically dependent software services [2]. The control and disclosure requirements originating from multiple compliance sources create auditing demands for SOAs.

One of the key requirements of a generic compliance management approach is that it should be sustainable throughout the business process (BP) lifecycle [1]. Compliance management should be considered from the early stages of business process design, thus achieving compliance-by-design, which must be further integrated with dynamic monitoring of the running instances. The emphasis in this paper is on requirements applicable to design-time phase of the BP lifecycle, while the discussions on the requirements applicable to later phases (runtime and offline phases) are kept limited.

Founding the specification of business process compliance requirements on a formal logical language enables their automatic verification and analysis against business process specifications. However, the complexity of the formal language must not become an obstacle for the specification and for their validation. It is important to find an appropriate balance between expressiveness, formal foundation, and potential analysis methods [3].

In this paper, we conducted a comparative analysis between three languages that can be used as the basic building blocks of a comprehensive *Compliance Request Language (CRL)* for the formal specification of compliance requirements, focusing primarily on *design-time* verification. In particular, we consider two families of logics that have been used successfully in the literature, namely deontic and temporal families of logic. More specifically, we consider Formal Contract Language (FCL) [4] from the deontic logic family. On the other hand, we consider Linear Temporal Logic (LTL) and Computational Tree Logic (CTL) from the temporal logic family [5-6]. We applied these languages on the specification of a wide range of compliance requirements of two industrial case studies explored within the EU funded COMPAS research project [7]. Then a comparative analysis was conducted based on the capabilities and limitations of each language against a set of identified key features. The comparative analysis reflected that the decision on the selection of a particular formal language is context-dependent involving various factors including the nature, complexity and source of compliance requirements. However, based on the results of the comparative analysis, we can argue that the temporal logic has advantages over others with regard to the specification of regulatory constraints.

The rest of this paper is organized as follows: Section 2 highlights the key features that should be maintained by a comprehensive compliance request language. Section 3 presents a simplified motivating scenario used as the running example throughout this paper. Section 4 briefly describes the key concepts and rules of the formalisms analyzed and examine their capabilities to express compliance requirements from the running scenario. The comparative analysis is drawn in Section 5. Related work is summarized in Section 6. Finally, conclusions and ongoing work are highlighted in Section 7.

## 2 Required Features of a Compliance Request Language

In order to reveal the features that should be possessed by a language to be used for the formal specification of compliance requirements, we have analyzed [8] a wide range of compliance legislations and relevant frameworks such as Basel II, Sarbanes-Oxley, IFRS, FINRA (NASD/SEC), COSO, COBIT and OCEG. This set of regulations and frameworks constituted a faithful representation of the range of compliance requirements that can be found within compliance legislations. We also conducted case studies on industry processes [9] that are subject to various regulatory compliance requirements (also discussed in Section 3). Based on the findings, we have identified a set of features that should exist in CRL. These features can be summarized as follows:

- *Formality*: The CRL should be formal to pave the way for the application of associated automatic analysis, reasoning and verification tools and techniques.
- *Expressiveness*: The CRL should be expressive enough to be able to capture the intricate semantics of compliance requirements.
- *Usability*: The CRL should not be excessively complex to inhibit users to understand and use it.
- *Consistency checks*: Contradictions and conflicts might arise between compliance requirements particularly when they originate from different sources. It is desirable for the CRL to provide mechanisms to identify and resolve these inconsistencies.
- *Normalization*: This feature refers to cleaning-up of the requirements specification to identify and remove redundancies and to make implicit requirements explicit.
- *Declarativeness*: Compliance requirements are commonly normative and descriptive, indicating what needs to be done [1]. Therefore, declarative languages are more suited to their formal representation.
- *Generic*: Compliance requirements can be constraints on the control-flow (sequence and timing of activities), data (data validation and requirements), and resource perspectives (task allocation and data access rights). The CRL should enable the specification of the requirements regarding to these perspectives.
- *Symmetry*: This feature refers to the ability to annotate business process models with compliance requirements. The annotation helps users to understand the interplay between the two specifications.
- *Non-monotonicity*: A violation to a compliance rule is not necessarily an error. Non-monotonic rules are open to violation to a certain extent and under specific conditions. Depending on the rigidity of the rule, the process expert can decide on the type of the rule, and the exceptions under which a specific rule can be overridden and the priorities among them.
- *Intelligible feedback*: Indicating whether there is a violation to a specific rule is not sufficient. It is important to provide the user with guidance of why a violation occurs and how to resolve compliance deviations.
- *Real-time support*: The language should be able to express real-time requirements, which are likely to appear in compliance sources. For example: Activity A should occur within time period  $k$ .

### 3 Running Scenario

The loan approval scenario is one of the industry case scenarios explored within the EU funded COMPAS research project [9]. The general environment in which this particular scenario takes place is banking e-business applications. Taking into account the demands for strong regulation compliance schemes, such as Basel II, Sarbanes-Oxley (SOX), ISO 27000 and sometimes contradictory needs of the different stakeholders, such scenarios raise several interesting compliance requirements.

**Table 1** An excerpt of compliance requirements relevant to the case scenario.

ID	Compliance Requirement (Context/Process Specific Interpretation)	Compliance Source
R1	<i>Only Post-processing Clerk and Supervisor roles can access the "Credit Bureau service".</i>	- Internal Policy
R2	<i>Customer bank privilege check is segregated from credit worthiness check.</i>	- SOX Sec.404 - ISO 27002 - 10.1.3
R3	<i>If the loan request's credit exceeds 1 million EURO the Clerk Supervisor checks the credit worthiness of the customer. The lack of the supervisor check immediately creates a suspense file. In case of failure of the creation of a suspense file, the manager is notified by the system and Post-processing Clerk is allowed to do the check.</i>	- Internal Policy - SOX Sec.404
R4	<i>As a final control, the branch office Manager has to check whether the request is profitable and risks are acceptable before making the final approval.</i>	- SOX Sec.404.
R5	<i>If loan conditions are satisfied, the customer can check the status of her loan request infinitely often until the customer is notified.</i>	- Internal Policy
R6	<i>If credit worthiness check activity is performed, then there exists an activity 'evaluation of the loan risk' that should be performed by the manager.</i>	- Internal Policy
R7	<i>The Credit Broker can start a loan (approved by the customer), only if 5 workdays or more have elapsed since the loan approval form was sent.</i>	- SOX Sec.404

The brief flow of the process is as follows: Once a customer loan request is received, the *credit broker* checks if customer's banking privileges are suspended. If privileges are not suspended, the credit broker accesses the customer information and checks if all loan conditions are satisfied. Next, a loan threshold is calculated, and if the threshold amount is less than 1M Euros, the *post processing clerk* checks the credit worthiness of the customer by conducting the credit bureau service. Next, the *post processing clerk* initializes the form and approves the loan. If the threshold amount is greater than 1M Euros, the *clerk supervisor* is responsible for performing the same activities instead of the post processing clerk. Next, the manager evaluates the loan risk, after which she normally signs the loan form and sends the form to the customer to sign. Table 1 lists an excerpt of the compliance requirements that are relevant to the scenario. The first and second columns of the table give a unique ID and a brief description of the original compliance requirement as they are in sources, respectively. Third column gives case scenario specific interpretation of the compliance requirement (internalized compliance requirements). Finally, the fourth column refers to the associated compliance source(s) (e.g. a legislation document).

## 4 Formalisms under Consideration

The following sub-sections present a brief description of the basic concepts and rules (syntax) of the considered logical languages, and examine their applicability to represent compliance requirements of the running scenario as described in Table 1.

### 4.1 Linear Temporal Logic (LTL)

LTL [5], [6] is a logic used to formally specify temporal properties of software or hardware designs. In LTL, each state has one possible future and can be represented using linear state sequences, which corresponds to describing the behavior of a single execution of a system. The formulas in LTL take the form  $Af$ , where  $A$  is the universal path quantifier and  $f$  is a path formula. A path formula must contain only atomic propositions as its state sub-formulas. The formation rules of LTL formulas are as follows:

- $\top$  and  $\perp$  are formulas ( $\top$  represents tautology and  $\perp$  represents contradiction).
- If  $P \in AP$ , where  $AP$  is a non-empty set of atomic propositions, then  $P$  is a path formula.
- If  $f$  and  $g$  are path formulas, then  $\neg f$ ,  $f \vee g$ ,  $f \wedge g$ ,  $Xf$ ,  $Ff$ ,  $Gf$ ,  $f U g$ ,  $f W g$  are path formulas ( $\vee$  represents ‘or’ and  $\wedge$  represents ‘and’ operators):
  - $G$  (*always*) indicates that formula  $f$  must be true in all the states of the path.
  - $X$  (*next time*) indicates that the formula  $f$  is true in the second state of the path.
  - $F$  (*eventually*) indicates that formula  $f$  will be true at future state of the path.
  - $U$  (*until*) indicates that if at some future state the second formula  $g$  will be true, then, the formula  $f$  must be true in all the subsequent states within the path.
  - $W$  (*weak until*) represents the same semantics as *until*, however it is evaluated to true even if the second formula  $g$  never occurs (note that  $pWq \equiv G(p) \vee (p U q)$ ).

### 4.2 Computational Tree Logic (CTL)

CTL [5], [6] is also a logic used to formally specifying temporal properties of software or hardware designs. CTL differs from LTL in terms of their underlying model of time. As opposed to LTL, in *branching* temporal logics, each moment in time may split into various possible futures. Hence, the structure under which branching temporal logic formulas are interpreted is represented as infinite computational tree, which describes the behavior of the possible computations of a nondeterministic program [10]. A well-formed CTL formula over a set of atomic propositions  $AP = \{p, q, \dots\}$  (where  $A$  is the universal path quantifier) can be formed as follows (in BNF notations):

- $\varphi ::= \top \mid \perp \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid AX\varphi \mid EX\varphi \mid AF\varphi \mid EF\varphi \mid AG\varphi \mid EG\varphi \mid A[\varphi U \psi] \mid E[\varphi U \psi] \mid A[\varphi W \psi] \mid E[\varphi W \psi]$ , such that:
  - $\top$  represents tautology and  $\perp$  represents contradiction symbols.
  - $G, F, X, U, W$  are the temporal operators ‘always’, ‘eventually’, ‘next’, ‘until’ and ‘weak until’ as defined in Section 4.1.

- $A$  and  $E$  stand for the universal (for *All* paths) and existential (there *Exists* a path) quantifies, respectively.
- Each CTL operator should be a pair of symbols. The first symbol is a quantifier ( $A$  or  $E$ ), and the second symbol is a temporal operator.

### 4.3 Formal Contract Language

FCL is a combination of an efficient non-monotonic formalism (defeasible logic) and deontic logic of violations. The FCL language consists of two sets of atomic symbols: A finite set of literals (propositions) that represent state variables, and a finite set of events. The logical operators that are supported are as follows: (i) ‘;’ the sequence operator, (ii) ‘ $\wedge$ ’ conjunction operator and (iii) ‘ $\vee$ ’ disjunction operator. A rule in FCL is an expression of the form:  $r: A_1, A_2, \dots, A_n \vdash B$ , where  $r$  is the identification of the rule,  $A_1, A_2, \dots, A_n$  is the set of premises (propositions) and  $B$  is the conclusion of the rule. The rule is built from a finite set of atomic propositions, logical operators, and a set of deontic operators, which are; (i) Negation ( $\neg$ ), (ii) Obligation ( $O$ ), (iii) Permission ( $P$ ), and (iv) the Contrary to duty operator ( $\otimes$  or CTD). Contrary to duty operator is used to specify the violations and the obligations arise as a response to the violations. The rules are formed as follows:

- Each atomic proposition is a proposition.
- If  $P$  is an atomic proposition, then  $\neg P$  is a proposition
- If  $P$  is an atomic proposition, then  $OP$  is an obligation proposition,  $PP$  is a permission proposition. Obligations and permission propositions are deontic propositions.
- If  $P \vdash P_1 \otimes P_2 \otimes \dots \otimes P_n \otimes q$  is an FCL rule, where  $P_1, P_2, \dots, P_n$  are obligation propositions and  $q$  is a deontic proposition then  $P_1 \otimes P_2 \otimes \dots \otimes P_n \otimes q$  is a reparational chain. The *reparational chain* indicates that, if the primary obligation  $P_1$  is violated, its violation can be repaired by the secondary obligation  $P_2$  and if  $P_2$  cannot be satisfied then it can be repaired by the obligation  $P_3$ , and so on. The entire rule is evaluated to false, if none of the primary obligation, or any of the reparation deontic propositions (respecting their order) is satisfied.
- Prohibitions can be either represented as  $O\neg$  or  $\neg P$ .

### 4.4 Formal Specification of Compliance Requirements

This sub-section examines and compares how compliance requirements of the running scenario as described in Table 1 can be formalized in the three languages.

<b><i>RI</i></b>	
<b><i>LTL:</i></b>	$G(\text{CheckCreditWorthiness.Role}(\text{Role1}) \rightarrow G(\text{Role1} = \text{'PostProcessingClerk'} \vee \text{Role1} = \text{'Supervisor'})))$ (1)
<b><i>CTL:</i></b>	$AG(\text{CheckCreditWorthiness.Role}(\text{Role1}) \rightarrow AG(\text{Role1} = \text{'PostProcessingClerk'} \vee \text{Role1} = \text{'Supervisor'})))$ (2)

$$\begin{array}{l} \text{FCL: } \text{Role1} \neq \text{PostProcessingClerk}' \wedge \text{Role1} \neq \text{Supervisor}' \\ \quad \vdash O_{\text{Role1}} \neg \text{CheckCreditWorthiness} \end{array} \quad (3)$$

This compliance requirement can be represented in the three languages. In FCL, the semantics of the requirement can be captured by prohibiting any other role rather than *PostProcessingClerk* and *Supervisor* from performing *CheckCreditWorthiness* activity. You can notice that the LTL and CTL representations can be viewed as the contra-positive of the FCL representation ( $P \rightarrow Q \equiv \neg P \rightarrow \neg Q$ )

### R2

$$\begin{array}{l} \text{LTL: } G((\text{CheckCustomerBankPrivilege.Role}(\text{Role1}) \\ \quad \rightarrow G(\neg(\text{CheckCreditWorthiness.Role}(\text{Role1}))) \end{array} \quad (4)$$

$$\begin{array}{l} \text{CTL: } AG((\text{CheckCustomerBankPrivilege.Role}(\text{Role1}) \\ \quad \rightarrow AG(\neg(\text{CheckCreditWorthiness.Role}(\text{Role1}))) \end{array} \quad (5)$$

$$\begin{array}{l} \text{FCL: } \text{CheckCustomerBankPrivilege.Role}(\text{Role1}); \\ \quad \text{CheckCreditWorthiness} \vdash O_{\text{Role1}} \neg \text{CheckCreditWorthiness} \end{array} \quad (6)$$

This requirement represents the typical segregation of duties constraint and it can be represented in the three languages.

### R3

$$\begin{array}{l} \text{LTL: } \text{Can't be represented} \end{array} \quad (7)$$

$$\begin{array}{l} \text{CTL: } \text{Can't be represented} \end{array} \quad (8)$$

$$\begin{array}{l} \text{FCL: } \text{LoanAmount}(y) \geq 1M' \vdash O_{\text{supervisor}} \text{CheckCreditWorthiness} \\ \quad \otimes O_{\text{System}} \text{StoreSuspenseFile}(y) \otimes O_{\text{System}} \text{notifyManager} \\ \quad \wedge P_{\text{PostClerk}} \text{CheckCreditWorthiness} \end{array} \quad (9)$$

This requirement can be represented in FCL using the  $\otimes$  (CTD) operator. Besides, FCL supports the notion of permission, which is *not* supported in LTL or CTL (e.g.  $P_{\text{PostClerk}} \text{CheckCreditWorthiness}$ ). Although semantically unequal, the closest operator in temporal logic is the disjunction operator, however, it is commutative.

### R4

$$\begin{array}{l} \text{LTL: } \neg \text{SignOfficiallyContract } W \text{ (JudgeHighRisk} \wedge \text{approved} = \text{'Yes'})} \end{array} \quad (10)$$

$$\begin{array}{l} \text{CTL: } A(\neg \text{SignOfficiallyContract } W \text{ (JudgeHighRisk} \wedge \text{approved} = \text{'Yes'})) \end{array} \quad (11)$$

$$\begin{array}{l} \text{FCL: } \text{Can't be represented} \end{array} \quad (12)$$

This requirement can't be represented in FCL due to its lack of support to temporal operators.

### R5

$$\begin{array}{l} \text{LTL: } FG(\text{LoanConditions} = \text{'True'} \rightarrow ((GF(\text{CheckLoanStatus})) U \text{NotifyCustomer})) \end{array} \quad (13)$$

$$\begin{array}{l} \text{CTL: } \text{Can't be represented} \end{array} \quad (14)$$

$$\begin{array}{l} \text{FCL: } \text{Can't be represented} \end{array} \quad (15)$$

Neither CTL nor FCL can express the *weak fairness* property of  $R5$  (a constantly enabled event must occur infinitely often) [5], which is expressible in LTL. The same applies to the specification of *strong fairness* properties.

<b><math>R6</math></b>		
<i>LTL:</i>	<i>Can't be represented</i>	(16)
<i>CTL:</i>	$AG(\text{CheckCreditWorthiness} \rightarrow EF(\text{JudgeHighRisk}))$	(17)
<i>FCL:</i>	<i>Can't be represented</i>	(18)

This requirement can only be expressed in CTL due to its support to the existential quantifier.

<b><math>R7</math></b>		
<i>MTL</i>	$G(\text{SendLoanContract} \rightarrow F_{\geq 5}(\text{PerformLoanSettlement.Role('CreditBroker')}))$	(19)
<i>TCTL</i>	$AG(\text{SendLoanContract} \rightarrow AF_{\geq 5}(\text{PerformLoanSettlement.Role('CreditBroker')}))$	(20)
<i>FCL</i>	$\text{SendContract: } t \vdash O_{\text{CreditBroker}} \text{ PerformLoanSettlement: } k \wedge k \geq t+5$	(21)

Requirement  $R7$  is not expressible in LTL or CTL due to their lack of support to real-time requirements. Some extensions to LTL and CTL have been proposed to incorporate real-time dimension. For example, Metrical Temporal Logic (MTL) [11] extends LTL with real-time dimension. In MTL, temporal operators can be annotated with a temporal expression  $I$  expressing a specific time interval as shown in the MTL representation of  $R7$  (e.g.  $F_{\geq 5}$ ). Timed CTL (TCTL) [12] extends CTL with real-time dimension exactly the same way as MTL extends LTL. Temporal dimension is also incorporated to FCL as proposed in [4], such that all propositions can be time-stamped. If we can conclude  $P$  at time  $t$ , written as  $P:t$ , then  $P$  is true for all  $t' > t$ , until an event occurs that terminates the validity of  $P$ .

## 5 Comparative Analysis between LTL, CTL and FCL

Table 2 summarizes the results of the comparative analysis, which highlights the strengths and limitations of the three languages. The degree of support is denoted by: '+', indicating that the feature is satisfied, '-', indicating that the feature is not satisfied; and '±', indicating that the support is partial.

Some of these results can be generalized to the whole families of Deontic logic and Temporal Logic. For example, FCL, CTL and LTL possess limitations in terms of *usability*. This result can be generalized to the whole families of Deontic and Temporal Logic. The complexity of logical languages represents one of the main obstacles of utilizing the sophisticated reasoning and analysis tools associated with these languages. FCL, LTL and CTL have different expressive powers. For example, the notion of permission is not expressible in LTL and CTL, while fairness properties are not expressible in FCL and CTL; on the other hand, existential properties are not expressible in LTL and FCL. Deontic and Temporal families of logic are declarative by nature. Furthermore, FCL provides a mechanism for consistency checks by the means of the *superiority relation* of the *defeasible logic* [13], yet this result can't be generalized to the Deontic Logic family (denoted by '?' in Table 2). Temporal Logic



family doesn't provide any support for checking consistency among logical formulas. The normalization metric is met by FCL as it provides a technique for cleaning up the specification and to identify and remove redundancies [4].

**Table 2:** Comparative Analyses of Compliance Request Languages

	LTL/ MTL	CTL/ TCTL	Temporal Logic	FCL	Deontic Logic
1- Formality	+	+	+	+	+
2- Usability	-	-	-	-	-
3- Expressiveness	±	±	?	±	?
4- Declarativeness	+	+	+	+	+
5- Consistency Checks	-	-	-	+	?
6- Non-Monotonicity	±	-	-	+	?
7- Generic	±	±	?	±	?
8- Symmetricity	-	-	?	±	?
9- Normalization	-	-	-	+	?
10- Intelligible feedback	+	±	?	-	-
11- Real-time Support	+	+	?	+	?

Non-monotonic requirements can be expressed in FCL by means of the *superiority relation*. On the other hand, rules in temporal logic are monotonic by nature. In FCL, by exploiting the results in [1], business process models can be visually annotated by compliance requirements using the notion of *control tags*. However, with symmetricity we mean the actual augmentation of business process models with compliance requirements (thus the support for this feature is marked as '±' for FCL). Model-checkers are used with temporal logic for automatic compliance verification [6]. As concluded in [10], it is usually possible with LTL to provide the *counterexample tracing* facility that helps experts to resolve a compliance violation, thus providing the user with intelligent feedback. The support to this feature is limited for CTL. In [14], we propose a comprehensive 'root-cause analysis' to reason about design-time compliance violations to provide the user with guidelines as remedies to resolve compliance deviations, which is based on LTL. The support by Deontic logic family to this criterion is limited.

Several extensions to LTL and CTL have been proposed to incorporate real-time dimension (e.g. MTL [11] and TCTL [12]). Real-time dimension is also incorporated to FCL as proposed in [4]. Finally, a basic strength of LTL and temporal logic in general lies in its maturity and availability of sophisticated *verification tools* that have proven to be successful to verify complex systems [10].

Vardi provides an interesting comparison between LTL and CTL in [10]. Although, CTL and LTL correspond to two distinct views of time, and consequently LTL and CTL are expressively incomparable. However, from a practical point of view LTL is considered to be more expressive than CTL. Besides, LTL is considered to be more intuitive than CTL. The un-intuitiveness of CTL significantly reduces the usability of CTL-based formal verification tools. From a verification point of view, CTL is considered to be more difficult than LTL due to the branching nature of CTL. Furthermore, CTL does not provide support for compositional reasoning. The main advantage of CTL over LTL is its *computational complexity*. However, Vardi argues

that LTL is a more powerful logic and CTL's advantage in terms of computational complexity is valid under rare circumstances in real life applications. On the other side, the computational complexity of FCL is unknown (compliance verification is based on the *Idealness* notion as proposed in [4]).

## 6 Related Work

In [15], a comparison is conducted between three types of logics: (i) CL (Contract Language): Deontic logic, (ii) LTL and CTL: temporal logics and (iii) Communicating Sequential Processes (CSP): operational language, with respect to their expressiveness to represent three requirements emerging from a business contract. Although we agree with the conclusion highlighting CL's power to represent the business contract under consideration, we diverge with the argument that states LTL's lack of support to some fairness properties. Although our main focus in this study is on regulatory compliance, the comparative analysis conducted in this paper is more generic and considers an extensive list of comparison criteria in addition to the *expressiveness* metric.

It is also of relevance here to summarize various key studies that utilize temporal and deontic logic for design-time compliance verification. Authors in [16] propose a static-compliance checking framework that includes various model transformations. Compliance requirements are based on LTL formulas. Next, NuSMV2 model checker is used to check the compliance. The study in [17] utilizes  $\pi$ -Logic to formally represent compliance requirements; while BP models are abstractly modeled using BP-Calculus. If business and compliance specifications are compliant, an equivalent BPEL program can be automatically generated from the abstract BP-calculus representations. The study in [18] utilizes past LTL (PLTL), where properties about the past can be represented. The study in [19] utilizes patterns to overcome the complexity of temporal logic focusing on runtime monitoring. The study in [20] utilizes Dwyer's patterns for the verification of service compositions. In [21], real-time temporal object logic is proposed for the formal specification of compliance requirements based on a pre-defined domain ontology. In [22], we use LTL and propose a framework for augmenting business processes with reusable fragments to ensure process compliance to the relevant requirements by design.

We have to point out that there is a third class of languages that can be used for the formal specification of compliance requirements grounded on XML, e.g. the XML Service Request Language (XSRL) [23] and the PROPOLS language [20]. Since XML-based approaches are founded on temporal logic, then they are subsumed by LTL and CTL, subsequently, they are not considered in our analytical study.

Key studies that have utilized Deontic logic can be summarized as follows: the work in [4] has provided the foundations of the FCL (Formal Contract Language) language, focusing on business partner contracts. In addition, in [13], an automatic transformation of business contracts represented in FCL to RuleML is proposed for runtime monitoring. In [1], FCL is used to express other types of compliance requirements emerging from legislation and regulatory bodies. In [24], the PENELOPE (Process Entailment from the Elicitation of Obligations and

PERmissions) language was proposed. PENELOPE is a language to express temporal deontic assignments considering only obligations and permissions.

## 7 Conclusion and Outlook

An important question that might arise in the field of compliance management is: “How compliance requirements can be formally specified to enable the application of automatic analysis and reasoning technique for their verification?” Temporal and deontic families of logic have been successfully utilized in the literature as the formal foundation of compliance requirements. In this paper, we report a comparative analysis between LTL, CTL and FCL. The comparison surfaces the strengths and limitations of each language with respect to a set of identified features. Some of these conclusions can be generalized to the whole family of temporal or deontic logic. The decision on the use of a particular formal language is context-dependent that should be based on the nature, complexity and source of compliance requirements. However, based on the overall findings of the comparative analysis as well as the relevant literature and the current practice, we argue that temporal logic has advantages over other formalisms under consideration when formal specification of regulatory compliance requirements is concerned. An important strength in temporal logic is its maturity and its sophisticated tool support.

It should also be noted that the identified comparison criteria are not equally important. For example, the support of temporal logic to the *intelligible feedback* and *sophisticated tool support metrics* is significant. We also agree with Vardi’s argument in [10] that LTL is a more powerful logic. CTL\* is the logic that combines the expressive power of LTL and CTL, however, its computational complexity is *2PTime-Complete*.

An interesting ongoing research direction is to resolve the main problems of LTL that have surfaced from the comparative analysis. In particular, developing a graphical language tool based on recurring property patterns [25] relevant to the compliance context would address the usability metric. Besides, providing efficient solutions to support the specification of non-monotonic rules in LTL, as well as normalization and consistency checking are other areas for future research. Finally, analyzing and investigating these languages on the basis of the support they provide not only for design-time verification but also for runtime monitoring -hence, integrating these two phases and providing a lifetime compliance management support - is an important ongoing research direction.

**Acknowledgments.** We express our sincere thanks to PricewaterhouseCoopers (Netherlands) and Thales Services SAS (France) for their effort in providing and participating in the case studies and scenarios, and their valuable contributions.

## References

1. Sadiq, S., Governatori, G., Naimiri, K.: Modeling Control Objectives for Business Process Compliance. 10th BPM'07, pp. 149-164, Australia (2007)
2. Papazoglou, M., Traverso, P., Dustdar, S., Leymann, F.: Service-Oriented Computing: State of the Art and Research Challenges. *Computer* 40, 38-45 (2007)
3. Thao, L., Goser, K., Rinderle-Ma, S., Dadam, P.: Compliance of Semantic Constraints- A Requirements Analysis for Process Management Systems. 1st GRCIS'08 workshop, pp. 41-45, France (2008)
4. Governatori, G., Milosevic, Z., Sadiq, S.: Compliance Checking Between Business Processes and Business Contracts. 10th EDOC'06, pp. 221-232, Hong Kong (2006)
5. Pnueli, A.: The Temporal Logic of Programs. 18th IEEE Symposium on Foundations of Computer Science, pp. 46-57, Providence (1977)
6. Clarke, E., Grumberg, J., Peled, D.: Model Checking. MIT Press, Cambridge (2000)
7. COMPAS official web site – Project description, <http://www.compas-ict.eu/project.php>
8. COMPAS Project, D 2.1, State-of-the-Art in the Field of Compliance Languages. (2008)
9. COMPAS Project, D 6.1, Use Case, Metrics and Case Study Definition. (2008)
10. Vardi, M.: Branching vs. Linear Time: Final Showdown. 7th TACAS'01, pp. 1-22, Italy (2001)
11. Alur, R., Henzinger, T.: Real-time Logics: Complexity and Expressiveness. *Information and Computation* 104, 35-77 (1993)
12. Alur, R.: Techniques for Automatic Verification of Real-time Systems. vol. Ph.D. thesis. Stanford University (1991)
13. Governatori, G.: Representing Business Contracts in RuleML. *International Journal of Cooperative Information Systems* (2005)
14. Elgammal, A., Turetken, O., van den Heuvel, W., Papazoglou, M.: Root-Cause Analysis of Design-time Compliance Violations on the basis of Property Patterns. 8th ICSOC'10, USA (2010)
15. Fenech, S., Okika, J., Pace, G., Ravn, A., Schneider, G.: On the Specification of Full Contracts. 6th FESCA'09 workshop, UK (2009)
16. Liu, Y., Muller, S., Xu, K.: A Static Compliance-Checking Framework for Business Process Models. *IBM Systems Journal* 46, (2007)
17. Abouzaid, F., Mullins, J.: A Calculus for Generation, Verification, and Refinement of BPEL Specifications. 3rd WWV'07 workshop, pp. 43-68, Italy (2007)
18. Awad, A., Decker, G., Weske, M.: Efficient Compliance Checking using BPMN-Q and Temporal Logic. BPM'08, pp. 326-341. (2008)
19. Namiri, K., Stojanovic, N.: Pattern-based Design and Validation of Business Process Compliance. 15<sup>th</sup> CoopIS'07, 59-76, Portugal (2007)
20. Yu, J., Manh, T., Han, J., Jin, Y.: Pattern-Based Property Specification and Verification for Service Composition. 7th WISE06, pp. 156-168, China (2006)
21. Giblin, C., Liu, A., Muller, S., B., P., Zhou, X.: Regulations Expressed As Logical Models. 18th JURIX'05, pp. 37-48, Belgium (2005)
22. Schumm, D., Turetken, O., Kokash, N., Elgammal, A., Leymann, F., van den Heuvel, W.: Business Process Compliance through Reusable Units of Compliant Processes. 1st ESW'10 workshop. LNCS, Austria (2010)
23. Lazovik, A., Aiello, M., Papazoglou, M.: Planning and Monitoring the Execution of Web Services Requests. *International Journal on Digital Libraries* 6, 235-246 (2006)
24. Goedertier, S., Vanthienen, J.: Designing Compliant Business Processes with Obligations and Permissions. 9<sup>th</sup> BPM'06, pp. 5-14, Austria (2006)
25. Dwyer, M., Avrunin, G., Corbett, J.: Property Specification Patterns for Finite-State Verification. 2nd International workshop on Formal Methods on Software Practice, pp. 7-15, USA (1998).